

ZUSAMMENFASSUNG

L. 02.11.15
Simone
Guggiari

ALPHABETE, SPRACHEN, AUFGABEN

Alphabete: Σ • endliche, nichtleere Menge
• elements called "Buchstaben, Zeichen, Symbole"

Word: $\{0,1\}^*$

Set: $\{a,b,\dots,z\}$

Wörter: • endliche (ev. leere) Folge von Buchstaben auf Σ

λ = leere Wort $|\lambda| = 0$
every alphabet contains λ
can be built from every alphabet

$|w|$ = length of w (# Buchstaben)

Σ^* : all buildable words over Σ , length ≥ 0

Σ^+ $\xrightarrow{\quad}$ Σ^* $\xrightarrow{\quad}$ Σ^+

$\Sigma^+ = \Sigma^* \setminus \{\lambda\}$

words buildable over Σ , length $\leq i$:

$$= \sum_{k=0}^i |\Sigma|^k = \frac{|\Sigma|^{i+1} - 1}{|\Sigma| - 1}$$

Concatenation: Abbildung $K: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$

$$K(x,y) = x \cdot y = xy, \quad x,y \in \Sigma^*$$

$$\text{associative: } K(x, K(y,z)) = x \cdot (y \cdot z) = (x \cdot y) \cdot z = K(K(x,y), z)$$

$$|xy| = |x \cdot y| = |x| + |y|$$

$$i^{\text{th}} \text{ iteration: } \begin{matrix} x^0 = \lambda \\ x^1 = x \end{matrix} \quad \begin{matrix} x^2 = x \cdot x \\ x^3 = x \cdot x \cdot x \end{matrix}$$

$|x|_a$ = # of occurrences of symbol a in x

Teilwörter, Präfixe, Suffixe

v Teilwort of $w \iff \exists x,y \in \Sigma^*: w = xvy$

Präfix

$\exists x$

$$w = vx$$

Suffix

"

$$w = xv$$

echtes Teilw./s

if $v \neq \lambda$ (or $|x| \geq 1$)
 $v \neq w$

Kanonsche Ordnung

$\Sigma = \{s_1, s_2, \dots, s_n\}$, with total order $s_1 < s_2 < \dots < s_n$

therefore the order on words is $u < v \iff |u| < |v|$ or

(shorter or comes first)

$$|u| = |v| \text{ and } u = xs; o^i, \quad v = xs; v^j, \quad i < j$$

Sprachen / Languages $\uparrow$ all habits around here

• language over Σ is Teilmenge of Σ^*

• complement of $L = \Sigma^* \setminus L$

$$\begin{matrix} \bullet L_\emptyset = \emptyset & \text{is empty language} & (0 \text{ words}) \\ \bullet L_\lambda = \{\lambda\} & \text{is language with empty word} & (1 \text{ word}) \end{matrix}$$

Konkatenation:

$$L_1 \cdot L_2 = L_1 L_2 = \{vw \mid v \in L_1, w \in L_2\}$$

Operations: $L^0 = L_\lambda$ $L^i = \bigcup_{j \in \mathbb{N}} L^j$ (union of all languages)
 $L^{i+1} = L^i \cdot L$ $L^+ = L \cdot L^*$ $\rightarrow$ called Kleene's star

$\Delta \mid L_1 \cdot L_2 \neq L_1 \mid L_2$ since words in set could be created

following relations hold true:

$$\bullet \Sigma^i = \{w \in \Sigma^* \mid |w| = i\}$$

$$\bullet L_\emptyset \cdot L = L_\emptyset = \emptyset = L \cdot L_\emptyset \quad (\text{can't combine any word})$$

$$\bullet L_\lambda \cdot L = L \quad (\text{combine with nothing})$$

$$\bullet L_1 L_2 \cup L_1 L_3 = L_1 (L_2 \cup L_3) \quad \rightarrow \text{union der single elem}$$

$$\bullet L_1 (L_2 \cap L_3) \subseteq L_1 L_2 \cap L_1 L_3 \quad \rightarrow \text{see lemma p. 21}$$

$\rightarrow$ could be bigger, since some word present in both could have been created in different ways

$$(\{a\}^*, \{ab\}^*)^* \neq \{a,b\}^* \quad \text{as left can't create } \{b\}^*$$

Homomorphism: $\Sigma_1^* \rightarrow \Sigma_2^*$ is a function h with properties

$$\bullet h(\lambda) = \lambda$$

$$\bullet \forall u,v \in \Sigma_1^* \quad h(uv) = h(u) \cdot h(v)$$

"it maps words correctly"

$\Rightarrow$ enough to see/determine mapping $h(a) \forall a \in \Sigma_1$

Algorithmische Probleme

Algorithmus: function $A: \Sigma_1^* \rightarrow \Sigma_2^*$ input $\xrightarrow{\text{alg computes}}$ output

output of A , input $x = A(x)$

alg A, B äquivalent $\iff \forall x: A(x) = B(x) \quad (A \sim B)$

Entscheidungsprobleme: $(\Sigma, L): \forall \text{ input } x \in \Sigma^* \text{ decide whether } x \in L$

$$A(x) = \begin{cases} 1 & \iff x \in L \\ 0 & \text{otherwise} \end{cases}$$

this is an algorithm that solves this problem

Δ languages for which such alg exists are called **RECURSIVE**

Transformationen

Alg A computes transformation (func) $f: \Sigma^* \rightarrow \Gamma^*$ if

$$\forall x \in \Sigma^*: A(x) = f(x)$$

erica prob are special cases of this, since "die berechnen die charakteristische Funktion einer Sprache"

Relationsprobleme

Alg A computes relation $R \subseteq \Sigma^* \times \Gamma^*$ if

$$\forall x \in \Sigma^* \text{ for which } \exists y \in \Gamma^*: (x,y) \in R \text{ gilt } (x, A(x)) \in R$$

it suffices to find one y of potentially infinitely many

Optimierungsprobleme $\rightarrow$ see p. 27

Aufzählungsprobleme

A zählt eine Sprache L über Σ auf, if given $n \in \mathbb{N}$ it outputs the first n elements of the language in kanonische Ordnung

Lexicographic order

for Cartesian product $(a,b) \leq (a',b') \iff a < a' \text{ or}$

in dictionary: $a_1 a_2 \dots a_n < b_1 b_2 \dots b_j$ if $(a = a' \text{ and } b \leq b')$

the first letter a_i in which they differ is $a_i < b_i$

doesn't matter length

Binäre Kodierung von Zahlen

$n \rightarrow \text{Bin}(n)$

symbols for bin code of $n = \lceil \log_2(n+1) \rceil$

0, 1
00, 01, 10, 11
000, 001, 010, 011, ...

0
1 2
3 4 5 6

word $x = \text{Bin}(n)$ is the $(n+1)$ th word of Σ_{bin} in numerical order (there is x)

? explain

Kolmogorov-Komplexität

Algo generates $x \in \Sigma^*$, if it outputs x on input λ

$K(x)$ of $x \in \Sigma_{\text{bin}}^*$ = bin length of shortest program (in pascal, but language doesn't matter) that generates x

$\forall x \in \Sigma_{\text{bin}}^* : K(x) \leq |x| + d$ (exists such d)

d : specifies prefix & suffix length + output code

begin $w, k(x)$
end

$K(n \in \mathbb{N}) = K(\text{Bin}(n))$

zufällig: (random, nicht komprimierbar)

$x \in \Sigma_{\text{bin}}^* : K(x) \geq |x|$
 $n \in \mathbb{N} : K(n) \geq \lceil \log_2(n+1) \rceil - 1$

length of bin. representation

random numbers of each length
 $\forall n \in \mathbb{N} \exists w_n \in \Sigma_{\text{bin}}^n : K(w_n) \geq n$

+ beweis
must know why there are

L language over Σ_{bin} , z_n is n th word in L (ken-ordnung)
if algo solves entscheid. probl. (Σ_{bin}, L) , then it holds
 $K(z_n) \leq \lceil \log_2(n+1) \rceil + c$

why?

$\text{Prim}(n) = \#$ of primes $\leq n$
primzahlssatz: $\lim_{n \rightarrow \infty} \frac{\text{Prim}(n)}{n} = \frac{1}{\ln(n)}$

→ distribution of primes

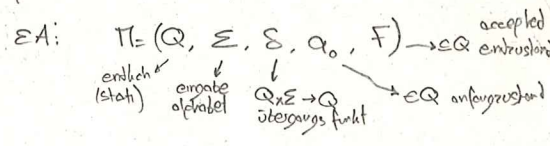
$n_1, n_2, \dots$ unendlich, $K(n_i) \geq \frac{\log_2 n_i}{2}$
then is $Q = \{q_i : i \in \mathbb{N}\}$ unendlich.

$\triangleright q_i$: die grösste primzahl with $q_i | n_i$

2. ENDLICHE AUTOMATEN

Def: programs that compute without memory.

deterministisch EA:



Konfiguration

current state + remain of input word

$(q, w) \in Q \times \Sigma^*$

(q_0, x) = startkonfiguration

$Q \times \Sigma^*$ = endkonfiguration → accepted konf $\in F \times \Sigma^*$

Schritt

konfig $\xrightarrow{\text{schritt}}$ konfig $R_{\Pi} \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$

$(q, w) R_{\Pi} (p, x) \iff (w = ax, a \in \Sigma) \wedge (\delta(q, a) = p)$

Berechnung

$C = C_0 C_1 \dots C_n$ = folge von Konfigurationen

$\forall i \in \{0, \dots, n-1\} : C_i R_{\Pi} C_{i+1}$

C is computation of Π on input $x : C_0 = \{q_0, x\}$
 $C_n \in F \times \Sigma^*$

accepted language

if $C_n \in F \times \Sigma^*$, then x is accepted by Π .

$L(\Pi) = \{w \in \Sigma^* \mid \Pi \text{ accepts } w\}$

powerset $P(\Sigma^*)$

$L(EA) = \{L(\Pi) \mid \Pi \text{ is EA}\} \subseteq P(\Sigma^*)$

is the class of languages, accepted by EA. called REGULAR

transit. hülle (closure)

reflexive hülle, transitive hülle, reflexive transitive hülle

$\vdash_{\Pi}^*$ of schrittrelation $\vdash_{\Pi}$

$(q, w) R_{\Pi}^*(p, v) \iff (q = p \wedge w = v) \text{ or } \exists k \in \mathbb{N} : w = a_1 a_2 \dots a_k u, a_i \in \Sigma$

$\exists r_1, \dots, r_{k-1} \in Q : (q, w) \vdash_{\Pi} (r_1, a_1 \dots a_k u) \vdash_{\Pi} \dots \vdash_{\Pi} (r_{k-1}, a_{k-1} a_k u) \vdash_{\Pi} (p, v)$

$\hat{\delta}$ = application of δ to $\hat{p}$

Zustandsklassen

$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

to each word, its endzustand is assigned

$\hat{\delta}(q, \lambda) = q$

$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

EA partitions Σ^* in $|Q|$ disjoint teilengen
it is equivalence relation (refl, sym, trans)

$K[L_P] = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) = p\}$

- $K[L_P] \cap K[L_Q] = \emptyset, p \neq q$ (disjunkt)
- $x \sim y \iff \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$
- $L(\Pi) = \bigcup_{p \in F} K[L_P]$ (union of accepted states)

Komplexitätsbeweise

$\geq \epsilon : L(\Pi) = L$

use induction by following:

- 1) beweise für $\lambda, x \in \Sigma^*$
- 2) annahme gilt $x \in \Sigma^*$, beweise für $y \in \Sigma^*$

use this relation

$\hat{\delta}(q_0, y) = \hat{\delta}(q_0, za) = \delta(\hat{\delta}(q_0, z), a)$

induktion → fallunterscheidung

2) SIMULATIONEN

$$\begin{aligned} \triangleright \Pi_1 &= (Q_1, \Sigma, \delta_1, q_{01}, F_1) \\ \Pi_2 &= (Q_2, \Sigma, \delta_2, q_{02}, F_2) \end{aligned}$$

for each $a \in \{0, 1, \lambda\} \exists \pi$ with

$$L(\Pi) = L(\Pi_1) \cup L(\Pi_2)$$

to build it:

$$\Pi = (Q, \Sigma, \delta, q_0, F)$$

$$Q = Q_1 \cup Q_2$$

$$q_0 = (q_{01}, q_{02})$$

$$\forall p \in Q, q \in Q, a \in \Sigma: \delta(p, q, a) = (\delta_1(p, a), \delta_2(q, a))$$

$$F = \begin{cases} F_1 \times Q_2 \cup F_2 \times Q_1 \\ F_1 \times (Q_2 \setminus F_2) \\ F_2 \times (Q_1 \setminus F_1) \end{cases} \quad \begin{aligned} & \delta = \cup \\ & = \cap \\ & = \setminus \end{aligned}$$

+beivers

PROOF OF NONREGULARITY

→ show $\nexists$ EA $\Pi: L(\Pi) = L$ which means $L \notin L(EA)$

Var. 1 - Lemma 3.3

$$\triangleright A = (Q, \Sigma, \delta, q_0, F) \text{ EA}$$

$$x, y \in \Sigma^*, x \neq y \quad x, y \in K[L]$$

it holds $\forall z \in \Sigma^*, \exists r \in Q: xz, yz \in K[L]$

(this follows from δ and δ)

$$\text{therefore } xz \in L(A) \iff yz \in L(A)$$

example:

indirect für $L = \{0^n 1^n \mid n \in \mathbb{N}\}$

$$\triangleright A = (Q, \Sigma, \delta, q_0, F) \text{ EA with } L(A) = L$$

consider the words $0, 0^2, 0^3, \dots, 0^{|Q|+1}$ since there are more words than states, it must follow

$$\exists i < j \in \{1, 2, \dots, |Q|+1\}: \hat{\delta}(q_0, 0^i) = \hat{\delta}(q_0, 0^j)$$

according to the lemma it must hold $\forall z \in \Sigma^*$

$$0^i z \in L \iff 0^j z \in L$$

but if you choose $z = 1^i$, then $0^i 1^i \in L$ but $0^j 1^i \notin L$ since $i < j$.

widerspruch. $\square$

Var. 2 - Pumping lemma

$\triangleright L$ regular. then $\exists n \in \mathbb{N}$, s.d. every word w with $|w| \geq n_0$ is decomposable as $w = xyz$

$$|xy| \leq n_0 \rightarrow \text{do we assign } n_0, \text{ or } n_0$$

$$|x| \geq 1$$

$$\{x^k y z \mid k \in \mathbb{N}\} \subseteq L \vee \{x^k y z \mid k \in \mathbb{N}\} \cap L = \emptyset$$

see weaker rules

example

indirect für $L = \{0^n 1^n \mid n \in \mathbb{N}\}$

$$\triangleright A = (Q, \Sigma, \delta, q_0, F) \text{ EA with } L(A) = L$$

consider the word $w = 0^{|Q|+1} 1^{|Q|+1}$ according to the lemma it must exist decomposition $w = xyz$ with all 3 properties.

Since $|xy| < |Q|$, it has to hold $x = 0^i$ for some i . now it has to hold

$w' = x^k y z \in L$. But this doesn't, since the language condition is not satisfied therefore contradiction $\square$

Δ in some cases, start from $\notin L$ and show that $\in L$

Var. 3 - Satz 3.1 (Kleene's method)

$\triangleright L \subseteq \Sigma^*$ regular

let's define prefix language

$$L_x = \{y \in \Sigma^* \mid xy \in L\} \quad \forall x \in \Sigma^*. \text{ let } w \text{ be the } n^{\text{th}} \text{ word from } L_x, \text{ then}$$

$$\exists c: K(w) \leq \lceil \log_2(n+1) \rceil + c \quad (\text{since it's enough } n \text{ to generate } w)$$

example: indirect für $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ assume it's regular.

let's build

$$L_{0^n} = \{y \mid 0^n y \in L\} = \{1^n \mid n \in \mathbb{N}\} = \{0^i 1^{n-i} \mid i \in \mathbb{N}\}$$

jetzt ist 1^n the 1st word, so it must hold

$$K(1^n) \leq \log_2(1+n) + c = 1+c$$

since we can build infinitely many L_{0^n} for every n , we have shown that there are infinite words with complexity $1+c$. but this isn't possible, since only a finite number of programs have length $1+c$.

NICHTDETERMINISMUS

NEA: from certain configuration you can choose one of more actions.

$$\text{formal } \Pi = (Q, \Sigma, \delta, q_0, F)$$

$$\delta: Q \times \Sigma \rightarrow P(Q) \quad \text{powerset, multiple choices}$$

Schritt:

$$(q, w) \vdash_{\Pi} (p, x) \iff (w = ax, a \in \Sigma) \wedge p \in \delta(q, a)$$

→ more states possible from each configuration

Berechnung:

same as EA

one of following properties holds:

$$\bullet C_n \in Q \times \Sigma^*$$

$$\bullet C_n = (q, ax), \delta(q, a) = \emptyset$$

L is accepted, if $\forall w \in L$

$\exists$ akzeptierende Berechnung

$$\hat{\delta}: Q \times \Sigma^* \rightarrow P(Q)$$

$$\bullet \forall q \in Q: \hat{\delta}(q, \lambda) = \{q\}$$

is a set

$$\bullet \hat{\delta}(q, wa) = \{p \mid \exists r \in \hat{\delta}(q, w): p \in \delta(r, a)\}$$

means since

$$= \bigcup_{r \in \hat{\delta}(q, w)} \delta(r, a)$$

$$\begin{aligned} L(\Pi) &= \{w \in \Sigma^* \mid \exists p \in F: (q_0, w) \vdash_{\Pi}^* (p, \lambda)\} \\ &= \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\} \end{aligned}$$

to check if NEA accepts w , you must consider all calculations.

you can use the Berechnungsbaum $B_{\Pi}(x)$

Berechnungsbaum (177)

Zurückführung auf EA - Potenzmengenkonstruktion

$$\Pi: \text{NEA}$$

$$A: \text{EA}$$

$$\forall \Pi \exists A: L(\Pi) = L(A)$$

build EA from NEA:

$$\Pi = (Q, \Sigma, \delta, q_0, F)$$

$$A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$$

$$\bullet Q_A = \{ \langle P \rangle \mid P \subseteq Q \}$$

→ all elements of the powerset

$$\bullet q_{0A} = \langle \{q_0\} \rangle$$

$$\bullet F_A = \{ \langle P \rangle \mid P \subseteq Q \wedge P \cap F \neq \emptyset \}$$

$$\bullet \delta_A: Q_A \times \Sigma \rightarrow Q_A \text{ is defined } \forall \langle P \rangle \in Q_A$$

$$\delta(\langle P \rangle, a) = \langle \bigcup_{p \in P} \delta(p, a) \rangle$$

Komplexitätsbeweis

3. GRAMMATIKEN

Def: mechanisms to generate words (which compose language)
→ Set of languages generated this way = Recursive languages
↳ decidable (computable)

$$G = (\Sigma_N, \Sigma_T, P, S)$$

$\Sigma_N \cap \Sigma_T = \emptyset$

$\Sigma_N, \Sigma_T \rightarrow$ words from Σ
 Σ (alphabet)

non-terminal (A, B, C, X, Y, Z)
terminal (a, b, c, ..., 0, 1, 2, ...)

rules

slot $\in \Sigma_N$

$$\Sigma = \Sigma_N \cup \Sigma_T$$

disjunkt

$$P: \Sigma^* \Sigma_N \Sigma^* \times \Sigma^*$$
$$(\alpha, \beta) \in P \leftrightarrow \alpha \rightarrow \beta$$
$$X \rightarrow \alpha X \beta$$

ableitung + erzeugen
 $\gamma \Rightarrow_G \delta$ (δ derivable in 1 step)
 $\exists w_1, w_2 \in \Sigma^*. \exists (\alpha, \beta) \in P: \gamma = w_1 \alpha w_2, \delta = w_1 \beta w_2$

derivable: $\gamma \Rightarrow_G^* \delta$ if $\gamma = \delta$ or $\exists w_1, w_2, \dots, w_n: \gamma = w_1 \alpha w_n, \delta = w_n \beta w_1, \alpha \Rightarrow_G^* \beta$

if $S \Rightarrow_G^* \delta$, then it is generated
 $L(G) = \{x \in \Sigma_T^* \mid S \Rightarrow_G^* x\}$

$\Rightarrow_G^i \beta$ if it takes exactly i steps to derive it

* see beweis $L(G) = L = \{w \in \Sigma_T^* \mid |w|_a = |w|_b\}$
 $G = \{ \{S\}, \{a, b\}, P, S \}$
 $P = \{ S \rightarrow \lambda, S \rightarrow SS, S \rightarrow aSb, S \rightarrow bSa \}$
use word length

Chomsky-hierarchy

- 4 classes of grammars
- Typ-0: general, unconditional grammar
- Typ-1: kontext-sensitiv grammar: rule $aXa \rightarrow b$ depends on the context
 $\Rightarrow \forall (\alpha, \beta) \in P: |\alpha| \leq |\beta|$ (can't substitute with shorter string)
- Typ-2: kontext-frei grammar
 $\Rightarrow \forall (\alpha, \beta) \in P: \alpha \in \Sigma_N^* \wedge \beta \in \Sigma^*$ (rules have form $X \rightarrow \beta$)
- Typ-3: regular grammar
 $\Rightarrow \forall (\alpha, \beta) \in P: \alpha \in \Sigma_N \wedge \beta \in (\Sigma_T^* \cdot \Sigma_N) \cup \Sigma_T^*$
rules have form $X \rightarrow uY$ or $X \rightarrow u$

a language L is of type $i \in \{0, 1, 2, 3\}$ if $\exists$ grammar $G: L(G) = L$
 $\mathcal{L}_i$: family of languages of class i

$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$

Regular grammar

$\mathcal{L}_3$ contains all endliche sprachen. (can build grammar that derives every word from S in one step)

$\mathcal{L}_3$ supports union (vereinigung)

$\triangleright L, L' \in \mathcal{L}_3$. therefore $\exists G, G'$ with $L(G) = L, L(G') = L'$

remove non-terminals, such that $\Sigma_N \cap \Sigma_N' = \emptyset$.

now build $G'' = (\Sigma_N'', \Sigma_T'', P'', S'')$ with $L(G'') = L(G) \cup L(G')$

$$\begin{cases} \Sigma_N'' = \{S''\} \cup \Sigma_N \cup \Sigma_N' \\ \Sigma_T'' = \Sigma_T \cup \Sigma_T' \\ P'' = \{S'' \rightarrow S, S'' \rightarrow S'\} \cup P \cup P' \end{cases} \rightarrow \text{bew} \subseteq$$

$\mathcal{L}_3$ supports concatenation and * (kleene star)

construction: (after renaming)

$$\begin{cases} \Sigma_N'' = \Sigma_N \cup \Sigma_N' \\ \Sigma_T'' = \Sigma_T \cup \Sigma_T' \\ S'' = S \end{cases}$$
$$P'' = P' \cup (P \cap (\Sigma_N \times \Sigma_T^* \Sigma_N)) \cup \{A \rightarrow wS' \mid (A \rightarrow w) \in P\}$$

$X \rightarrow uY$
↳ take all the new terminal ones

↳ can continue after word is finished

Normierte reguläre grammatik

reg. gram. is normalized (normalisiert) if all rules have one of the following forms:

- $S \rightarrow \lambda$ (S starts equal)
- $A \rightarrow a, A \in \Sigma_N, a \in \Sigma_T$ (only 1 Σ_T !)
- $B \rightarrow bC, B, C \in \Sigma_N, b \in \Sigma_T$

you can convert each reg. gram. to it's normalized version using rules:

- 1) remove KETTENREGEL: $X \rightarrow Y, Y \rightarrow aa \rightarrow X \rightarrow aa$
- 2) remove $A \rightarrow \lambda$ (A not start): $X \rightarrow aY, Y \rightarrow \lambda \rightarrow X \rightarrow a$
- 3) remove multi-form: $X \rightarrow abcY \rightarrow X \rightarrow aY_1, Y_1 \rightarrow bY_2, Y_2 \rightarrow c$

Bezug zum DEA
 $\mathcal{L}_3 = L(EA)$

beweis: $A = (Q_A, \Sigma_A, \delta, q_0, F_A)$ $G = (\Sigma_N, \Sigma_T, P, S)$
 $L(A) = L(G)$

build grammar $\Sigma_N = Q_A, \Sigma_T = \Sigma_A, S = q_0$
 $P = \{ p \rightarrow aq \mid a \in \Sigma_A, p, q \in Q_A: \delta(p, a) = q \} \cup \{ f \rightarrow \lambda \mid f \in F_A \}$
↳ b finish

build EA $\Rightarrow$ normierte gram. build NEA

$\Pi: (\Sigma_N \cup \{q_f\}, \Sigma_T, \delta, S, F)$

$$F = \{ q_f \}$$
$$S \rightarrow \lambda \notin P$$
$$q_f \in P$$
$$\delta(q_f, a) = \emptyset \quad \forall a \in \Sigma_T$$

$X \in \Sigma(X, a)$ falls $X \rightarrow aY \in P, X, Y \in \Sigma_N, a \in \Sigma_T$
 $P \in \Sigma(X, a)$ falls $X \rightarrow a \in P, X = "$

$\mathcal{L}_3$ supports kompement

$$L \in \Sigma^*, L \in \mathcal{L}_3 \Rightarrow (\Sigma^* \setminus L) \in \mathcal{L}_3$$

to build reg. gram. that accepts kompement of language generated by another reg. grammar:

reg. gram. $G \xrightarrow{\text{normalize}} G' \xrightarrow{\text{build}} \text{NEA} \xrightarrow{\text{polenzuerge konstruieren}} \text{EA} \xrightarrow{\text{build}} \text{EA}'$ (accepts kompement!)

build $\rightarrow$ reg. grammar for EA'

Church'sche These

TM equivalent to algorithm: $\mathcal{L}_R$ (recursive language class = computable) $\equiv$ language, algorithm, recursive

Δ nicht beweisbar $\Rightarrow$ Axiom of T.I.

program

Auer is program that decides whether bitstring $x \in \Sigma_{\text{bool}}^*$ is $\text{Kod}(\pi)$

$x \in \Sigma_{\text{bool}}^*$; $\forall i \in \mathbb{N}^+$ is x coding of i^{th} TM if

- $x \in \text{Kod}(\pi)$ for $\pi(\pi)$

• set $\{y \in \Sigma_{\text{bool}}^* \mid y < x \text{ in lex-ordering}\}$ contains (i-1) Kod of TM

i^{th} TM is π_i has order:

NON-DETERMINISTIC TM

for each arg $\rightarrow$ multiple actions available

$\text{NTM} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ (same as TM)

only δ changes:

$\delta: (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, N, R\})$ it is a powerset non
 $\delta(p, \phi) \subseteq \{(q, \phi, X) \mid q \in Q, X \in \{L, N, R\}\}$ (it's a property) don't overrule/so left of ϕ

konfig, schritt exactly like TM

$\rightarrow$ can accept or reject

$\rightarrow$ can now have ≥ 0 possibilities to transition, as output of δ

accepted language: same as TM

Berechnungsbaum

TM, x with NTM π computing on input $x \in \Sigma^*$: potentiell unendlich directed graph

• each node = konfiguration

$\rightarrow$ root = startkonfig ($q_0 \phi x$)

• edge = schritt, from node C to its children

$\Rightarrow$ π is NTM, then $\exists A$ TM with $L(\pi) = L(A)$

if π doesn't compute infinitely, then also A stops for each input.

TM KODIERUNG

describe all TM with alphabet $\{0, 1, \#\}$

$\pi = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ with $Q = \{q_0, q_1, \dots, q_m, q_{\text{acc}}, q_{\text{rej}}\}$
 $\Gamma = \{A_1, A_2, \dots, A_r\}$

let's define symbols

$Q \begin{cases} C(q_i) = 10^{i+1} 1 \\ C(q_{\text{acc}}) = 10^{m+2} 1 \\ C(q_{\text{rej}}) = 10^{m+3} 1 \end{cases}$ $\Gamma \begin{cases} C(A_j) = 110^j 11, j=1, 2, \dots, r \end{cases}$
 none $\begin{cases} C(L) = 111 000 11 \\ C(N) = 111 0 11 \\ C(R) = 111 00 11 \end{cases}$

and transitions

$C(\text{trans}) =$

$C(\delta(p, A_i)) = (q, A_j, \alpha) = \# C(p) C(A_i) C(q) C(A_j) C(\alpha) \#$

to code the machine:

give # states, # symbols of Γ , transitions

$C(\pi) = \# 0^{m+3} \# 0^r \# \# C(\text{trans } 1) \# C(\text{trans } 2) \# \dots$

to map $\{0, 1, \#\}$ to Σ_{bool} , use homeomorphism $h: \{0, 1, \#\}^* \rightarrow \Sigma_{\text{bool}}^*$

$h(\#) = 01$ $h(1) = 11$ $h(0) = 00$

$\text{Kod}(\pi) = h(C(\pi))$

the set of $\text{Kod}(\pi)$ is $\text{Kod TM} = \{\text{Kod}(\pi) \mid \pi \in \text{TM}\}$

Nummerierung:

$\mathbb{N}^{\{0,1\}} \times \mathbb{N}^{\{0,1\}}$

$f(a, b) = \binom{a+b-1}{2} + b$ it's bijective

ECHENBARKEIT - computability

$A \leq B$ iff $\exists f: A \rightarrow B$ injective (no collisions)

$$|A| \leq |B| \wedge |B| \leq |A| \iff |A| = |B|$$

countable if $|A|$ finite or $|A| = |\mathbb{N}|$. it follows $\forall \Sigma^*$ countable (abzählbar) (canonical order)

know these sets are countable:

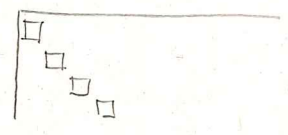
$$\mathbb{N}^+ \times \mathbb{N}^+ \quad \bullet \mathbb{Q}^+ \quad \bullet \text{Kod } \Pi$$

$[0,1] \in \mathbb{R}$ is not countable (proof using cantor diagonalization method)

consider infinite-length representation of all numbers $\in [0,1]$, listed.

now build number that differs from every other in a single digit (the one on the diagonal)

will never find a place in the list.



diagonalization

$P(\Sigma_{\text{bool}}^*)$ of all languages isn't countable.

proof: injective $f: [0,1] \rightarrow P(\Sigma_{\text{bool}}^*)$, showing $|P(\Sigma_{\text{bool}}^*)| \geq |[0,1]|$

this using diagonal language

$$L_{\text{diag}} = \{w \in \Sigma_{\text{bool}}^* \mid w = w_i, i \in \mathbb{N}^+ \wedge \Pi_i \text{ does not accept } w_i\}$$

follows that $\exists$ non-recursive (computable) languages

$$|\text{Kod } \Pi| < |P(\Sigma_{\text{bool}}^*)|$$

$\hookrightarrow \# \text{ of } \Pi$

reduction

$$L_1 \leq L_2 \iff L_1 \in \Sigma_{\text{bool}}^*, L_2 \in \Sigma_{\text{bool}}^*$$

$$L_1 \leq L_2 \implies L_1 \in \Sigma_{\text{bool}}^* \implies \exists \Pi_1, \Pi_2 \text{ that always stops with } L(\Pi) = L_1$$

$$\leq_{\text{RE}} L_2 \implies \text{convert to output}$$

$$\Pi \text{ computing } f_{\Pi}: \Sigma_1^* \rightarrow \Sigma_2^* \forall x \in \Sigma_1^* \iff f_{\Pi}(x) \in L_2$$

$$\leq_{\text{RE}}^L, L^C \leq_{\text{RE}} L \text{ in alg, } q_{\text{acc}} \text{ and } q_{\text{rej}} \text{ are inverted}$$

universal language

$$L_U = \{\text{Kod}(\Pi) \# w \mid w \in \Sigma_{\text{bool}}^*, \Pi \text{ accepts } w\}$$

$\exists \Pi_U$ with $L(U) = L_U$, which simulates Π on w (accepts/rejects/no-loop)

$\Rightarrow$ you can't know without simulation whether Π accepts w

$$\Delta L_{\text{diag}} \leq_{\text{RE}} L_U$$

(using L_U you can decide whether a word is in L_{diag} or not)

$$L \in \Sigma_{\text{bool}}^* \implies L^C \notin \Sigma_{\text{bool}}^*$$

otherwise you would always know for every input after a finite amount of steps where it belongs which would make it total in Σ_{bool}^*

$$\leq_{\text{RE}}^L: \text{show } L_{\text{diag}}^C \leq_{\text{RE}} L_U \text{ (we know } L_{\text{diag}}^C \notin \Sigma_{\text{bool}}^*)$$

halteproblem

is the entscheidungsproblem $(\{0,1\}^*, L_H)$

$$L_H = \{\text{Kod}(\Pi) \# x \mid x \in \{0,1\}^*, \Pi \text{ stops on } x\}$$

trivial $L_H \in \Sigma_{\text{bool}}^* \implies$ proof? show $L_H \notin \Sigma_{\text{bool}}^*$ using $L_U \leq_{\text{RE}} L_H$ and $L_U \notin \Sigma_{\text{bool}}^*$

$\hookrightarrow$ if it stops, we know it's accepted

empty language

$$L_{\text{empty}} = \{\text{Kod}(\Pi) \mid L(\Pi) = \emptyset\} \implies \Pi \text{ doesn't accept any input}$$

$$L_{\text{empty}}^C = \{x \in \Sigma_{\text{bool}}^* \mid (x \neq \text{Kod}(\Pi) \text{ for all } \Pi) \text{ or } (x = \text{Kod}(\Pi) \text{ and } L(\Pi) \neq \emptyset)\}$$

don't forget second argument!

$$L_{\text{empty}}^C \notin \Sigma_{\text{bool}}^*: \text{ use for proof } L_U \leq_{\text{RE}} L_{\text{empty}}^C \text{ try it!}$$

Satz von Rice

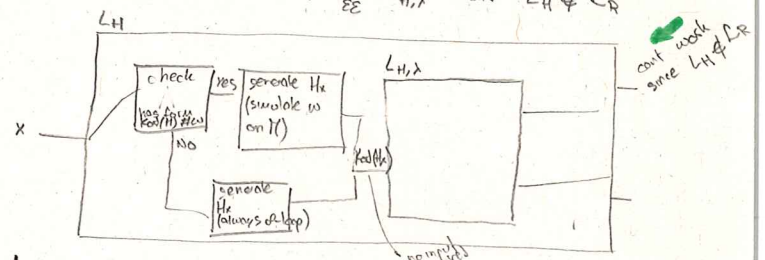
satz: all SNEUT is not decidable

$L \subseteq \{\text{Kod}(\Pi) \mid \Pi \text{ is a } \Pi\}$ is called SNEUT (semantisch nichttriviales Entscheidungsproblem über Π)

- $L \neq \emptyset$ ($\exists \Pi_1 \Pi_2$ s.d. $\text{Kod}(\Pi_1) \in L$)
- $L \neq \text{Kod } \Pi$ ($\exists \Pi_1 \Pi_2$ s.d. $\text{Kod}(\Pi_2) \notin L$)
- for Π_1, A, B , if $L(A) = L(B) \implies (\text{Kod}(A) \in L \iff \text{Kod}(B) \in L)$
 $\Rightarrow$ it is undecidable to determine whether the bty. of an arbitrary Π lies in L

specific halteproblem: $L_{H,x} = \{\text{Kod}(\Pi) \mid \Pi \text{ stops on } x\}$

$\Sigma\Sigma: L_{H,x} \notin \Sigma_{\text{bool}}^* : \text{ use } L_H \leq_{\text{RE}} L_{H,x} \text{ and } L_H \notin \Sigma_{\text{bool}}^*$



satz: $\forall$ SNEUT unentscheidbar ist ($\notin \Sigma_{\text{bool}}^*$)

beuers: $\forall$ entscheidungsproblem L entweder $L_{H,x} \in \Sigma_{\text{bool}}^* \text{ or } L_{H,x} \in \Sigma_{\text{bool}}^C$

equivalent language

$$L_{\text{acc}} = \{\text{Kod}(\Pi) \# \text{Kod}(N) \mid L(\Pi) = L(N)\} \notin \Sigma_{\text{bool}}^*$$

Kolmogorov complexity method

problem: $x \in \Sigma_{\text{bool}}^* \implies K(x)$ is $\notin \Sigma_{\text{bool}}^*$ (not algorithmisch entscheidbar)

beuers: $\exists$ program that generates first word x_n for which $K(x_n) \geq n$ on input n

its complexity is $K(x_n) \leq \log_2 n + c$ (input is n)

but $n \leq K(x_n) \leq \log_2 n + c$ holds for a finite number of numbers

$\Rightarrow$ the existence of an alg (that would compute K of them)

with this method: if $L_H \in \Sigma_{\text{bool}}^*$, then $\exists$ Alg to compute $K(x) \forall x \in \Sigma_{\text{bool}}^*$ (it doesn't!)

$$\rightarrow \nexists \text{ Alg } A: A(n) = K(n)$$

5.5: Post'sche Korrespondenz problem?

6. KOMPLEXITÄTSTHEORIE

Komplexitätstheorie

Size of alphabet doesn't matter $\rightarrow$ if you increase its size, memory+time complexity decrease

π is (TM) that always stops $\sum_{input\ alphabet} x \in \Sigma^*$ $D = C_1 C_2 \dots C_k$ Berechnung (each is binary)

Zeit:

$Time_{\pi}(x) = k-1$

$Time_{\pi}(n) = \max \{ Time_{\pi}(x) \mid x \in \Sigma^n \}$

$\forall$ k-Band TM A (always stops) $\exists$ equivalent TM B s.d. $L(A) = L(B)$

$\rightarrow Time_B(n) \leq \frac{Time_A(n)}{2} + 2n$

$\rightarrow$ beweis!

Speicherkomplexität

π is k-band TM (always stops) $D = C_1 C_2 \dots C_k$

$C = (q, x, i, (u_k, i_k), \dots, (u_k, i_k))$ a configuration

$Space_{\pi}(C) = \max \{ |u_i| \mid i = 1 \dots k \}$ space compl. of config

$Space_{\pi}(x) = \max \{ Space_{\pi}(C_i) \mid i = 1 \dots \ell \}$ " of computation

$Space_{\pi}(n) = \max \{ Space_{\pi}(x) \mid x \in \Sigma^n \}$ " for input length

$\forall$ k-Band TM A (always stops) $\exists$ 1-Band TM B s.d. (more efficient)

$\rightarrow Space_B(n) \leq Space_A(n)$ (done by increasing $|E|$)

$\bullet \exists$ k-Band TM B with $L(A) = L(B)$

$Space_B(n) \leq \frac{Space_A(n)}{2} + 2$

$\rightarrow$ beweis (build $\Gamma_B = \Gamma_A \times \Gamma_A$ save 2 symbols pro cell (+ 4 + decoding bit))

Asymptotik

$N \rightarrow \mathbb{R}^+$

$\bullet O(f(n)) = \{ g(n) \mid \exists n_0, c \in \mathbb{N} : \forall n \geq n_0, g(n) \leq c \cdot f(n) \}$ not faster

$\bullet \Omega(f(n)) = \{ g(n) \mid \exists n_0, c \in \mathbb{N} : \forall n \geq n_0, g(n) \geq \frac{1}{c} \cdot f(n) \}$ at least so fast

$\bullet \Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$ same speed

$\bullet o(f(n)) = \{ g(n) \mid \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0 \}$ slower than

Komplexitätsschranken

$\rightarrow$ consider complexity of optimal alg for problem: not suitable

$\bullet \exists$ Entscheidungsproblem Z, s.d. $\forall$ TM A that decides Z $\exists$ TM B equivalent, with $Time_B(n) \leq \log_2(Time_A(n))$ for infinite n (can always improve)

Δ

$O(f(n))$ gives schranke $\rightarrow \exists$ TM A that decides L in $O(f(n))$ for language L

$\Omega(f(n))$ unless $\rightarrow \forall$ TM (doesn't exist) better one) holds $Time_A(n) \in \Omega(f(n))$

optimal: $\Omega(f(n))$ for L and TM A decides in $O(f(n))$ Zeitkomplexität

uniform vs log Kostenmass

Zeitloep: every op costs 1 much more appropriate

Speicherloep: # of used variables

Zeiten, he serves as non super growende variable

Zeitloep: sum of length of operators for op $\rightarrow$ sum that for all op

Speicherloep: sum of length of variable representations

COMPLEXITY CLASSES

for $f: N \rightarrow \mathbb{R}^+$

$TIME(f) = \{ L(B) \mid B \text{ is TM, } Time_B(n) \in O(f(n)) \}$

$SPACE(f) = \{ L(B) \mid B \text{ is TM, } Space_B(n) \in O(f(n)) \}$

$DLOG = SPACE(\log_2 n)$

$P = PTIME = \bigcup_{c \in \mathbb{N}} TIME(n^c)$

$PSPACE = \bigcup_{c \in \mathbb{N}} SPACE(n^c)$

$EXPTIME = \bigcup_{c \in \mathbb{N}} TIME(2^{n^c})$

$NTIME(f) = \{ L(B) \mid B \text{ is NTM, } Time_B(n) \in O(f(n)) \}$

$NSPACE(f) = \{ L(B) \mid B \text{ is NTM, } Space_B(n) \in O(f(n)) \}$

$NLOG = NSPACE(\log_2 n)$

$NP = NPTIME = \bigcup_{c \in \mathbb{N}} NTIME(n^c)$

$NPSPACE = \bigcup_{c \in \mathbb{N}} NSPACE(n^c)$

konstruierbarkeit von funktionen

$f: N \rightarrow N$

platzkonstruierbar: $\exists$ 1-Band TM π , s.d.

- $\forall n \in N: Space_{\pi}(n) \leq f(n)$
- on input 0^n , π generates $0^{f(n)}$ on the arbeitsband

zeitkonstruierbar: $\exists$ k-band TM π , s.d.

- $\forall n \in N: Time_{\pi}(n) \in O(f(n))$

lemmas:

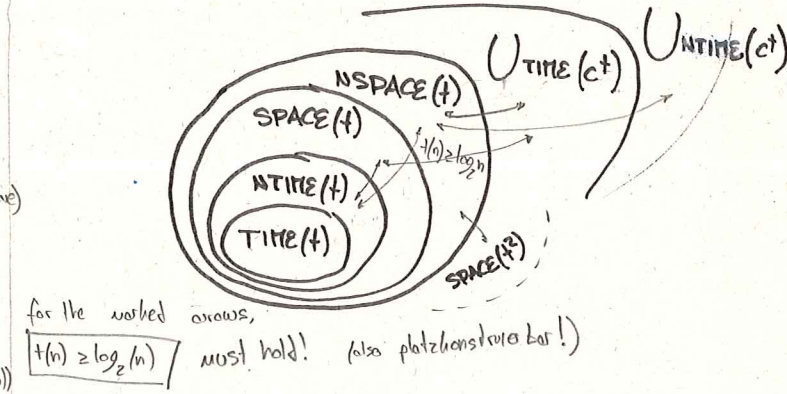
Δ platzkonst: π TM π , $Space_{\pi}(x) \leq f(|x|) \forall x \in L(\pi)$

then $\exists$ TM A equivalent, $L(A) = L(\pi)$ and $\forall n, Space_A(n) \leq f(n)$

Δ zeitkonst: π TM π , $Time_{\pi}(x) \leq f(|x|) \forall x \in L(\pi)$

then $\exists$ TM A equivalent, $L(A) = L(\pi)$ and $\forall n, Time_A(n) \in O(f(n))$

beweise: in A compute $f(n)$, then simulate π step and reject if it takes more than $f(n)$

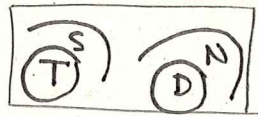
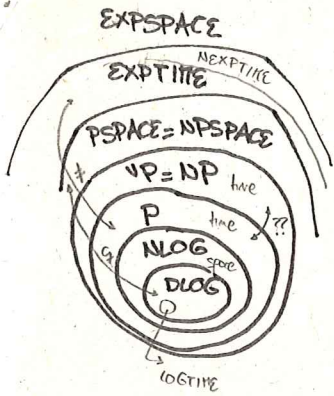


for the worked arrows, $t(n) \geq \log_2(n)$ must hold! (also platzkonstruierbar!)

Switch: $\bullet s(n) \geq \log_2 n$ platzkonstruierbar, then $NSPACE(s(n)) \subseteq SPACE(s(n)^2)$

$\rightarrow$ beweis case reachable

RELATIONS BETWEEN CLASSES



since $TIME(t(n))$ in time $Time_H(n)$ can't write more than $Time_H(n)$ fields, it holds

$$TIME(t(n)) \subseteq SPACE(t(n)) \quad (\text{can't be out})$$

$\forall$ platzkonstanterbar $f, f(n) \geq \log_2(n)$

$$SPACE(f(n)) \subseteq \bigcup_{c \in \mathbb{N}} TIME(c \cdot f(n))$$

times with inner config. (can't write more than $c \cdot \log_2(n)$) p178

$\triangleright f, g$ with $\begin{cases} g(n) \geq \log_2(n) \\ g(n) \text{ platzkonstanterbar} \end{cases}$

$\neg f(n) = o(g(n)) \Rightarrow SPACE(f) \not\subseteq SPACE(g)$

$\triangleright f, g$ with $\begin{cases} g(n) \text{ zeitkonstanterbar} \\ f(n) \cdot \log_2 f(n) = o(g(n)) \end{cases}$

$\Rightarrow TIME(f) \subseteq TIME(g)$ no beweis

Praktische Lösbarkeit $\equiv \exists$ polynomial alg (P)

- $O(n^{c \cdot 1})$ often can be optimized to $O(n^6)$
- independent from model of computation. models are poly-reducible, therefore they are the same

NONDETERMINISTIC COMPLEXITY

zeit: $\triangleright$ Π N-NTM. Σ input alphabet $x \in L(\Pi) \subseteq \Sigma^*$

$Time_H(x)$ = zeitkomplexität of shortest accepting computation on x (steps)

$Time_H(n) = \max \{ Time_H(x) \mid x \in L(\Pi) \cap \Sigma^n \}$

speicherkomplexität

$D = C_1 C_2 \dots C_\ell$ computation that accepts x

$Space_H(C_i)$ = speicherkomplexität of configuration C_i

$Space_H(D) = \max \{ Space_H(C_i) \mid i = 1 \dots \ell \}$

$Space_H(x) = \min \{ Space_H(D) \mid D \text{ is accepting computation of } x \}$

$Space_H(n) = \max \{ Space_H(x) \mid x \in L(\Pi) \cap \Sigma^n \}$

NP und BEWEISVERIFIKATION (P ? NP)

meaning: zeithorup of DET $\rightarrow$ true to find beweis
N-DET $\rightarrow$ true to check beweis

$\rightarrow$ NP-complete (Cook)

$SAT = \{ x \in \Sigma_{logic}^* \mid x \text{ is a satisfiable formula in CNF } \}$
 $\hookrightarrow \{ \neg, \wedge, \vee, \rightarrow, \perp, \top, \dots \}$ (or) and (or)

NTM "guesses" values and then verifies them deterministically

$SAT \in NP \rightarrow p199$

P-VERIFIZIERER

$L \subseteq \Sigma^*$ $p: \mathbb{N} \rightarrow \mathbb{N}$

Alg A is p -Verifizierer of L , $V(A)=L$ if A works on input $\Sigma^* x \Sigma_{bool}^*$

- $\forall$ input (w, x) : $Time_A(w, x) \in p(|w|)$
- if $w \in L$, $\exists x \in \Sigma_{bool}^*$ with $|x| \leq p(|w|)$ and $(w, x) \in L(A)$
 x is the beweis of statement " $w \in L$ "
- if $w \notin L$, $\forall x \in \Sigma_{bool}^*$ $(w, x) \notin L(A)$

$V(A) = L = \{ w \in \Sigma^* \mid \exists x \in \Sigma_{bool}^*, |x| \leq p(|w|), (w, x) \in L(A) \}$

if $p(n) \in O(n^k)$ A is polynomialverifizierer

All languages that have pol-verifizierer

$VP = \{ V(A) \mid A \text{ is pol-verifizierer} \}$

VP=NP

the question $P=NP$ means "is it easier to verify or to find a beweis?"

NP-COMPLETENESS

why we believe $P \neq NP$

- intuitive, beweisverifikation easier als beweiserrstellung
- many years of not finding poly-alg for NP-problems
- probably not due to incapability of researchers

reduction: $\triangleright L_1, L_2$ languages, $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$

L_1 polynomial reducible auf L_2 , $L_1 \leq_p L_2$ if $\exists$ poly-alg A that $\forall x \in \Sigma_1^*$ computes $A(x) \in \Sigma_2^*$, s.d.

$$x \in L_1 \iff A(x) \in L_2$$

NP-hard (schwer): $\forall$ languages $L' \in NP$: $L' \leq_p L$

L is NP-hard if

if L is also in NP, then it is called NP-COMplete

it follows

$$L \in P \wedge L \text{ NP-hard} \Rightarrow P=NP$$

proof: $\exists$ poly TM for L . show that $\exists$ poly NTM with $L(A) = U$ $\forall U \in NP \Rightarrow U \in P$

$P=NP$ or $\begin{matrix} NP \\ \text{complete} \\ P \end{matrix}$

how to prove $\in NP$?

MISSING PART

* see previous

cook: SAT is NP-complete

$$\text{SAT} \in \text{NP} = \text{NP}$$

$$L \in \text{NP} \leq_p \text{SAT}$$

means $\forall$ polynomial Π , $L(\Pi) \leq_p \text{SAT}$

L_1, L_2 languages. If $L_1 \leq_p L_2$ and L_1 NP-hard $\Rightarrow L_2$ NP-hard (use for reductions)

($\forall L' \in \text{NP}$, $L' \leq_p L_1$)
 levers: proof valid: assume L_2 NP-hard.
 $\exists L' \in \text{NP}$: $L' \not\leq_p L_2$. but since $L' \leq_p L_1$ and transitivity of $\leq_p$, we have $\leq$

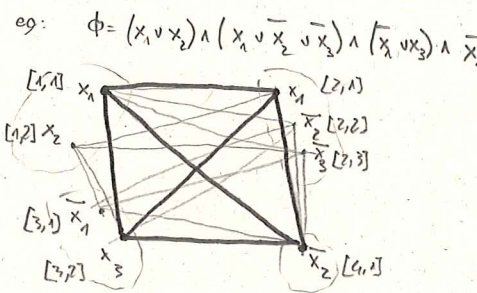
SAT = $\{\phi \mid \phi \text{ is satisfiable formula in CNF}\}$ (or) and/or

CLIQUE = $\{(G, k) \mid G \text{ undirected graph, contains } k\text{-clique}\}$ = $\{x \mid x \in E_{\text{clique}}\}$
 $\hookrightarrow$ connected subgraph

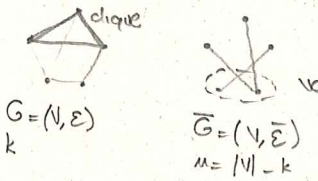
VC = $\{ \dots \}$ with vertex cover of max $|V| \leq k$

can be solved using
 $\text{SAT} \leq_p \text{CLIQUE}$
 $\hookrightarrow$ therefore NP-hard

connect every thing to what is not in same clause or discarding



CLIQUE $\leq_p$ VC
 $\geq_p$



3KNF: formula is in KNF, each clause max 3 literals

SAT $\leq_p$ 3SAT

$\hookrightarrow$ construction: $F = F_1 \wedge F_2 \wedge \dots \wedge F_m$

if F_i has < 4 literals, $C_i = F_i$

otherwise $F_i = z_1 \vee z_2 \vee \dots \vee z_k$, $k \geq 4$, $z_i \in \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$

use $\{x_1, \dots, x_n, q_{i,1}, q_{i,2}, \dots, q_{i,k-3}\}$ to build C_i

$$C_i = (z_1 \vee z_2 \vee q_{i,1}) \wedge (\bar{q}_{i,1} \vee z_3 \vee q_{i,2}) \wedge (\bar{q}_{i,2} \vee z_4 \vee q_{i,3}) \wedge \dots \wedge (\bar{q}_{i,k-4} \vee z_{k-2} \vee q_{i,k-3}) \wedge (\bar{q}_{i,k-3} \vee z_{k-1} \vee z_k)$$

then $C = \bigwedge C_i$. then $F \in \text{SAT} \Leftrightarrow C \in \text{3SAT}$
 both satisfiable or both unsatisfiable

3SAT $\leq_p$ VC?

levers: remember special assignment

OPTIMIERUNG

def: NPO class of optimization problems ($\sim \text{NP}$)

$$O = (\Sigma, \Sigma_0, L, \Pi, \text{cost}, \text{goal})$$

$L \in P$ (input, see if accepted)

$\exists$ poly $P_O : \forall x \in L, y \in \Pi(x), |y| \leq P_O(|x|)$
 (size of output poly to input)

$\exists$ alg $A \in P : \forall y \in \Sigma_0^*, \forall x \in \Sigma_1^*$ decides $y \in \Pi(x)$

cost is polynomial

MAX-SAT $\in \text{NPO}$
 MAX-CUT $\in \text{NPO}$ $\rightarrow$ bipartite, so #edges is max
 MIN-VC $\in \text{NPO}$
 MAX-CL $\rightarrow \text{NP-hard}$

def: PO class of opt.problem U , s.d. $U \in \text{NPO}$
 $\exists$ poly A, A , s.d. $\forall x \in L, A(x)$ optimal solution

Schnellwertsprache:

$$\text{Lang}_U = \{(x, a) \in L \times \Sigma_{\text{cost}}^* \mid \text{Opt}_U(x) \leq \text{Number}(a)\}$$

if goal = min
max

reduction to NP-hard problem

if opt.problem $U \in \text{PO}$, $\text{Lang}_U \in P$

$U \in \text{NPO}$. If U NP-hard $\wedge P \neq \text{NP}$, then $U \notin \text{PO}$

7. REGULAR EXPRESSIONS

they are language defining notation used in text-search applications $\Rightarrow$ declarative way to describe accepted strings

op: operations on languages that regex represent:

union: $L \cup R$

concatenation: LR

closure: L^*

notes: λ, ϕ are regex that represent $\{\lambda\}, \{\phi\}$ respectively (languages)

$$\Delta \phi^* = \{\lambda\} = \phi^0$$

$\phi^{i \geq 1}$ is empty

$a \in \Sigma \rightarrow a$ is regex, whose language is $L(a) = \{a\}$

variable, as L represents language

induction: E, F regex $L(E), L(F)$ language

$E + F$ is regex denoting union $L(E + F) = L(E) \cup L(F)$

$E \cdot F$ concatenation $L(E \cdot F) = L(E) \cdot L(F)$

E^* closure $L(E^*) = (L(E))^*$

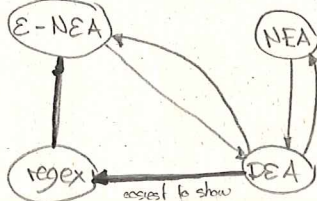
(E) same with parenthesis $L((E)) = L(E)$

optional symbol: concatenate $(\lambda + a)$

precedence: $\begin{matrix} * \\ \cdot \\ + \end{matrix}$ $()$ are above all

examples!

EA: both regex and EA define "regular languages"



$X \rightarrow Y$ every L in X is also in Y

graph is strongly connected - everything is same

DEA $\rightarrow$ regex

tricky - build expressions that define set of strings that label certain paths in the DEA transition

If $\exists$ EA A with $L(A) = L \Rightarrow \exists$ regex R with $L(R) = L$

$|Q| = n$

$R_{ij}^{(k)}$ = regex with language = strings that denote path from q_i to q_j without touching q_k with $i > k$ (ij don't have constraint)

start $k=0$, end $k=n$

$\rightarrow$ no intermediate states: direct $i \rightarrow j$
length 0 path, only node i (only if $i=j$) $\rightarrow$ string λ

no symbol: $R_{ij}^{(0)} = \phi$
1: $R_{ij}^{(1)} = a$
 k : $= a_1 a_2 + \dots + a_k$

induction: doesn't go through $k \rightarrow$ same as $R_{ij}^{(k-1)}$

does:



$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$

$\rightarrow + \nearrow \rightarrow^*$

end: The regex for the EA is the sum (union) of regex $R_{ij}^{(n)}$, with $q_i = q_{start}$, $q_j = q_{accept}$

simplification facts:

$$(\lambda + R)^* = R^*$$

$$\phi R = \phi = R \phi \quad (\text{annihilator})$$

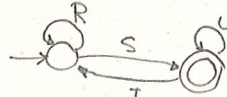
$$\phi + R = R + \phi = R \quad (\text{identity})$$

eliminating states

by eliminating state, ...

strategy: $\forall$ accepting state q_{acc} delete everything except q_{start}, q_{acc}

if $q_i \neq q_{acc}$, you are left with



accepting regex is

$$(R + S U^* T)^* S U^*$$

if start is accepting, perform elimination

